Summary

In the last 10-15 years, Computer Vision (CV) has evolved through multiple paradigms, from handcrafted feature engineering to deep learning models. This group assignment aims to explore these developments in practice by implementing and comparing key models across the last epochs in CV research.

Each group will work on one of the classical computer vision problems: **recognition, detection, segmentation, tracking or retrieval**. For the selected problem, you are expected to implement a pipeline to accommodate different models and evaluate their effectiveness on one suitable dataset.

Assignment submission

You should upload a single zip and PDF file containing:

- Codebase (zip):
    - a single script for the inference pipeline for the implemented models, which can be executed on CPU, and additional pipelines;
    - (if applicable) scripts or jupyter notebooks for model training;
    - README file with instructions on setting up libraries, datasets, and dependencies to run the files.
- Technical report (PDF). The report should be between 2000 and 3000 words, excluding references and an optional appendix.

Rubrics:

- 50% codebase:
    - 40%: The inference script is running and produces results similar to the ones reported. At least one model per epoch of CV can be used with this script.
    - 10% The README file is structured and written well. It contains all instructions to launch inference and training (if applicable) scripts.
- 50% technical report: clarity, structure, depth of analysis. The technical report should describe each student's contribution.

Notes:

- If graders cannot launch your scripts based on instructions from your README, you will be deducted more points.

Below, you can find instructions concerning the important parts of the assignment.

Implementations

For the implementation, you are free to choose any programming language of your choice. Nevertheless, we suggest using Python, since it provides a rich set of libraries and frameworks for data science (pandas, numpy, scipy), machine learning (scikit-learn), computer vision (cv2, PIL, torchvision) and deep learning (pytorch, huggingface, tensorflow).

Models

Each group must implement at least three models (one per student) from the two recent epochs of computer vision. To find relevant models for your problem, you are expected to conduct a brief literature review to select a pool of models to be implemented. Your work should contain at least one model from each epoch:

- **Early Deep Learning Models (2015-2019)**
  - Implement models based on deep learning architectures. Example architectures:
    - VGG16/VGG19
    - ResNet-18/ResNet-50
  - Train these models from scratch or using pre-trained weights.
- **Modern Deep Learning with Pre-trained and Fine-tuned Models (2020-now)**
  - Implement transfer learning using state-of-the-art model(s) of your choice. Example models can be based on:
    - EfficientNet, Vision Transformers (ViT), or Swin Transformer
  - Fine-tune these models on the dataset. Explore what fine-tuning strategies are feasible on your hardware.

Unified Pipeline

To ensure consistency, your group should build a single pipeline (e.g., a python script) that supports using all the above approaches for model prediction. General suggestions:

- The pipeline should be configurable to be executed with the specified model from the pool of the implemented models. You should aim to design a flexible integration of various feature extraction and prediction techniques.
- The pipeline should load and preprocess image(s) for feeding into model inference.
- You can optionally provide additional scripts or Jupyter notebooks used for model training.
- The pipeline, i.e. model inference, should run on CPU, whereas for training, you are free to use any hardware of your preference (CPU, laptop GPU, Google Colab).

Data and Experimental Setup

Each group must use one standard dataset for the assigned computer vision problem. Example datasets:

- Recognition (e.g., image classification): CIFAR (10 or 100), FashionMNIST
- Detection (e.g., object detection): COCO 2016 Detection, Pascal VOC
- Segmentation (e.g., semantic/instance segmentation): COCO 2016 Detection, Pascal VOC
- Tracking: TrackingNet (you can use a small subset for easier experimentation), KITTI, VisEvent (https://paperswithcode.com/task/object-tracking)
- Retrieval (e.g., text-to-image retrieval or content-based image retrieval): Flickr8k, Flickr30k, COCO

Note: some Deep Learning frameworks for Python (pytorch, huggingface, tensorflow) provide tools for downloading widely-used datasets (sometimes even processed or split into train/val/test) from their hubs. Feel free to use them! If you want to challenge yourself, you can, of course, re-implement these steps.

Make sure your experimental setup is technically sound and valid. Pay attention to how you split data into train, validation, and test sets to avoid possible leakages.

Evaluation Metrics

Conduct a comprehensive evaluation of your models and structurally present the obtained results:

- Identify valid performance metrics (e.g., classification accuracy, mAP, IoU, retrieval precision, etc.) for your tasks and use them for model evaluation.
- Computational efficiency and complexity (e.g., how much time does it take to train a model on your hardware, what are RAM or VRAM requirements, number of parameters)
- Additional qualitative evaluations obtained through concrete examples and visualization.

Report Guidelines

There is no fixed structure for the final technical report. As an example, it can contain the following sections:

Abstract (max 200 words)

- A summary of the study and key findings.

Introduction and Related Work (max 600 words)

- Importance and relevance of your CV problem.
- Problem statement and objectives.
- Brief discussion on the approaches to solve the problem from different epochs of CV and justification for the chosen models from each epoch.

Methodology and Experimental setup (max 800 words)

- Description of implemented models.
- Details on the dataset and preprocessing.
- Brief description of the implemented pipeline, utilized tools and hardware.

Results and Discussion (max 800 words)

- Performance comparison between models.
- Analysis of trade-offs (accuracy vs. computational efficiency and complexity).
- Qualitative results.
- Discussion of problem-specific challenges and insights.

Conclusions (max 200 words)

- Key takeaways and potential improvements.

Example: Content-based Image Retrieval

Dataset examples:

- CIFAR-10 or CIFAR-100

Models:

- Resnet-34:
    - Open-source Pytorch version pre-trained on ImageNet
    - Fine-tuning a couple of layers for the selected dataset (cross-entropy, metric learning).
- ViT – Vision Transformer pre-trained with a certain objective, e.g., MAE:
    - Open-source pre-trained version from Huggingface
    - Check whether it is possible to fine-tune this larger model on a small dataset using hardware you have access to.
- DINOv2 – state-of-the-art model based on vision transformers:
    - Open-source pre-trained version from Huggingface

Pipelines:

1. Inference script:
    - Iterate through the test set to generate representation (embedding) for each input image using the model provided from configs or command line arguments. Ensure modular implementation that allows you to switch between different retrieval models flexibly. If you fine-tune the model(s), please make sure you load the fine-tuned checkpoints in your inference script.
    - Compute similarity matrices between image embeddings (think of optimal similarity metric), and for each image, retrieve the most similar
    - Compare the implemented models:
        i. Using meaningful metrics. Discuss what retrieval metrics you can use and which ones fit the dataset specifics. Can you also comment on the statistical significance of the obtained results?
        ii. Show examples of the retrieved images. Highlight potential findings and shortcomings of the models.
2. Additional training scripts or notebooks:
    - If you fine-tune the models (e.g., Resnet in this case), you can do it in a separate script/notebook/colab that saves model checkpoints.